SciSpot

Paper # XXX

ABSTRACT

In this paper, we...

1 INTRODUCTION

Running parallel scientific applications, such as molecular dynamics (MD) simulations, on low-cost cloud transient resources.

The first "big" idea is that simulations are often bag of parallel tasks.

While there has been some past work that looks at running MPI applications on spot instances, our scope is much broader and considers how complete simulation pipelines can be run at low cost.

Spiel on transient instances. Increasingly popular resource allocation model that is being offered by all cloud providers. Very low cost compared to conventional cloud resources, often by up to 10x. However, can be frequently revoked. Thus failure is a common occurrence, and not a rare-event. This is especially challenging for MPI jobs because of its inability to tolerate failures.

However, our insight is that while protecting a *single* job against revocations can require elaborate checkpointing based approaches, we dont necessarily have to do that if we consider that most simulations are composed of a series of jobs that search over a parameter space, and that what is important is the total running time and cost of this entire series of jobs.

Thus, no single job is "special".

Another aspect of novelty is that past work on transient resources used EC2 pricing information to get failure probabilities. However, this is no longer an accurate method. We perform the first empirical study of google preemptible VMs and their performance and availability for HPC workloads.

Another fundamental question is what is a suitable metric in such cases. Conventionally, it is speedup. In the cloud, it is some combination of cost and running time.

2 BACKGROUND

2.1 Transient Computing

2.2 Parallel Scientific Applications in the Cloud

2.3 Molecular Dynamics Applications

Describe the kind of computation.

, ,
.

Scaling properties. Almost perfectly scalable with O(n) communication?

3 DESIGN

Scientific simulation applications consume a large amount of computational resources, and are often used in the context of *exploratory* research, where a large amount of jobs are run with different simulation parameters. This can be either a *parameter sweep*, where a large number of parameters need to be evaluated, or a *search* over a large parameter space for the “right” set of parameters that yield the desired model behavior.

In this paper, we look at the problem of running scientific simulations on *transient* computing resources in public clouds.

Past work has largely been focused on running parallel jobs (such as MPI) in the cloud. However, considering entire *job-groups* or ensembles of jobs presents new challenges and opportunities in timely, low-cost computation.

Our system, SciSpot, is a unified framework for running large job-groups that result from parameter exploration.

Input and some assumptions: We assume that the job-group consists of $J_1 \ldots J_N$, with each job evaluating a model on some parameter. The list of parameters to explore can either be generated apriori (as in the case of parameter sweeps), or be dynamically generated as in the case of a search.

In this section, we will look at how we address these challenges:

(1) How to select the right type of cloud server for an application?
(2) How to effectively run job-groups?

SciSpot’s key insight is considering an entire job-group can allow better and simpler optimizations that can be easily deployed.

3.1 Trade-offs in Server Selection

This presents us with many challenges in the cloud-deployment of these jobs.

Cloud providers offer multiple types of instances (VMs), with different hardware configuration (such as number of CPUs and memory size). The price of cloud servers is related to their hardware configuration, but it may not be strictly proportional to the hardware performance. For example, a VM with 32 CPUs may not be 32 times the cost of a single CPU VM.

For parallel and distributed applications, the type of servers selected has large implications on their performance. Consider the case of deploying an application on 8 8-core VMs vs. 16 4-core VMs. In both cases, the total number of CPU cores is the same. However, the larger number of VMs requires more communication between the application tasks, and thus may result in performance degradation. The performance of applications at different cluster configurations depends on their communication patterns and scaling properties.

Thus, when deploying applications on the cloud, one has to mindful of the cost and performance tradeoff. However, in the case of transient servers, the story does not stop here.

In addition to pricing differences, the transient availability of instances *also* differs by type. Because the availability of a transient VM is broadly determined by the overall supply and demand of the instances of that *particular* type, the "preemption rate" of VMs often depends on the type of the instance.

Thus, selecting a transient cloud server involves a complex tradeoff between the cost of servers, their performance, and the preemption-rate.

3.2 Server Selection Policy

SciSpot's server selection policy seeks to identify the best server type for a given job-group.

As stated in the previous subsection, the transient server selection problem is challenging because it involves balancing multiple optimization criteria: applications want low cost, low preemptions, and high performance.

Server selection based on application characteristics is a subject of a growing amount of recent work. These approaches often use micro benchmarks to gather performance data of cloud servers, and then use application performance models to determine suitable VMs for a specific application. Another class of approaches uses "black box" performance modeling, where the application's performance is modeled using a function of the resources, for example, by using linear regression.

In contrast to prior work, our server selection employs a "cold start" policy, and we do not run profiling or pilot jobs that can increase the overall running time and cost. Instead, we search for the "best" cluster configuration for jobs in a job-group, by exploring the cluster configuration space for the "optimum" server type that optimizes all the desired parameters: cost, running time, and revocation rate.

Thus, the first e jobs in the job group $J_1 \ldots J_e$ are the exploration jobs, run on different cluster configurations. We limit the total number of combinations to explore, by allowing users to submit an estimate of the total number of CPU cores that they desire for each job. This allows us to meet the user expectations in terms of performance and cost—whether the user expects us to spend a large amount of resources or not.

Thus, assuming that there are s different types of VM instances, the first s jobs are run on the s different types. Note that we use homogeneous clusters, since the performance of BSP programs in Heterogeneous environments can be degraded, and importantly, as we show, there are no performance or cost benefits to Heterogenity.

For each server type i, we calcuate the expected cost $E[C_i]$. $E[C_i] = n_i * c_i * E[T_i]$, where c_i is the price (per second) of the server, n_i is the number of servers of that type required to meet the core-count requirement. The expected running time of the job depends on two factors: the actual running time T, and the increase in running time due to preemptions. Each preemption is akin to a fail-stop failure, which requires an application to restart. Our system makes no assumptions about the fault-tolerance policies supported by the application. For example, some applications may be able to *checkpoint* their state periodically. In either case (checkpointing or not), there is some work lost due to revocations. For ease of exposition, we assume no checkpointing. We discuss checkpointing in the next section (or never?)

Expected running time: Let the running time without failure be T:

$$E[T_i] = T + P(\text{at least one failure}) * T/2 \tag{1}$$

For calcuating the probability of failure, we assume that the failure rate of an individual server of the type is p_i.

$$\begin{aligned} P(\text{at least one failure}) &= 1 - P(\text{no failure}) \quad (2) \\ &= 1 - (1 - p_i)^{n_i} \quad (3) \end{aligned}$$

Thus, we can see that if we select smaller VMs, we will require more of them (higher n_i), and this cluster configuration will have a larger probability of failure and thus higher running times and costs.

The probability of failure p_i depends on the type of server, and we use historically determined failure distributions. Roughly, if we assume exponentially distributed failures, then:

$$p_i = \frac{T}{\text{MTTF}_i} \tag{4}$$

Where MTTF is the mean time to failure of the server type, and T is the empirically determined job running time without failures (the best case).

In addition to searching over the servers, the effective number of servers is also dynamic in the case of transient environments due to preemptions. Thus, once we have found the appropriate server, we then explore the application's performance at smaller cluster sizes, which helps in the job-group policies that we discuss next.

3.3 Preemption-handling Policies

We run the remaining jobs on the right configuration that is determined through the server selection policy.

Our goal is to minimize costs given a deadline.

This determines two things: how many jobs should be run in parallel, and what to do upon a revocation.

The number of jobs in parallel determines the overall size of our cluster.

$$\text{number of parallel jobs} = \frac{N \cdot T}{\text{Deadline Duration}}$$

If a server is preempted, then the job running on it will cease to run. Our preemption handling policies then decide what to do:

(1) Restart the job on a smaller number of servers
(2) Replenish lost servers and restart job
(3) Discard job. This may be useful in case of parameter sweeps.

In addition, the user is also allowed to provide the fraction of jobs that are allowed to fail (η).

3.4 Checkpointing

Checkpointing requires the same number of servers, which may be tricky, whereas restarts can be on smaller number of nodes no problem.

3.5 Early Stopping

Based on the energy function, we can stop some simulations early. The early stopping criteria helps in minimizing the number of jobs run to completion.

We use this to proactively monitor jobs, as well as to decide whether to restart a job if it is preempted.

This can be a fairly substantial section

4 IMPLEMENTATION

Central controller.

Cloud APIs for launching the jobs.

Slurm.

5 EVALUATION

The contenders:

(1) Run every job on un-tuned on-demand instance (cost and running time)
(2) Run every job on nanohub/big-red-2 (running and waiting time)
(3) Run on transient, restart every time (cost)
(4) SciSpot with early stopping and job sacrificing

5.1 Preemption likelihood curves

5.2 Searching for the best cloud configuration

5.3 Total cost vs. running time graphs

6 RELATED WORK

6.1 Scientific applications on cloud

A classic survey is [6] [13]

Parameter sweep: [2]

Price optimizations for Scientific workflows in the cloud [4]

6.2 Transiency mitigation

[7] classic work on MPI and Spot. Uses checkpointing. Redundancy, but for what? User specified number of VMs. Does not do instance selection. BCLR for checkpointing.

MOre spot and MPI: [5]. FOcussed on bidding and checkpoint interval. But bidding doesnt matter.

[10] is early work for spot and MPI and

[9] a batch computing service

Heterogenity often used, but not useful in the context of MPI jobs. [8]

Selecting the best instance type, often for data analysis computations [1], and [12], and others like Ernest and Hemingway.

All the past work was on EC2 spot market with gang failures and independent markets [5, 7]. However this assumption has now changed, and failures can happen anytime. Our failure model is more general, and applies to both cases.

6.2.1 Fault-tolerance for MPI. [3] has a discussion of checkpointing frequency which is comprehensive.

Replication is another way [11]

6.2.2 Huge amount of work on bidding in HPC. [?] [?]

REFERENCES

[1] Alipourfard, O., and Yu, M. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. 15.

[2] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)* (May 2000), pp. 349–363.

[3] Dongarra, J., Herault, T., and Robert, Y. Fault tolerance techniques for high-performance computing. 66.

[4] GarÃŋ, Y., Monge, D. A., Mateos, C., and GarcÃŋa Garino, C. Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing* (Feb. 2019).

[5] Gong, Y., He, B., and Zhou, A. C. Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: checkpoints and replicated execution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15* (Austin, Texas, 2015), ACM Press, pp. 1–12.

[6] Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H. J. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems 22*, 6 (June 2011), 931–945.

[7] Marathe, A., Harris, R., Lowenthal, D., De Supinski, B. R., Rountree, B., and Schulz, M. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC* (2014), ACM.

[8] Sharma, P., Irwin, D., and Shenoy, P. Portfolio-driven resource management for transient cloud servers. In *Proceedings of ACM Measurement and Analysis of Computer Systems* (June 2017), vol. 1, p. 23.

[9] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC* (August 2015).

[10] Taifi, M., Shi, J. Y., and Khreishah, A. SpotMPI: A Framework for Auction-Based HPC Computing Using Amazon Spot Instances. In *Algorithms and Architectures for Parallel Processing*, Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7017. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 109–120.

[11] Walters, J. P., and Chaudhary, V. Replication-Based Fault Tolerance for MPI Applications. *IEEE Transactions on Parallel and Distributed Systems 20*, 7 (July 2009), 997–1010.

[12] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the *best* VM across multiple public clouds: a data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17* (Santa Clara, California, 2017), ACM Press, pp. 452–465.

[13] Zhai, Y., Liu, M., Zhai, J., Ma, X., and Chen, W. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications. In *State of the Practice Reports on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.